Who we are

» Nathan Byers
  » Database Analyst at Indiana University (IU) School of Medicine

» Kali Frost
  » Research Associate at IU School of Public Health

» Eric Bailey (*in absentia*)
  » Web Application Developer at IU School of Medicine

» Former employees at the Indiana Department of Environmental Management

» Experienced data analysts and R programmers
Who we are

- R enthusiasts
- Follow along at
  https://ebailey78.shinyapps.io/epaToxicsPresentation
What this is

- R presentation using ioslides
- Source code is on GitHub
- This is entirely reproducible
- Comic credit: https://xkcd.com/242/
Training Outline

- What is R and RStudio
- R basics
- The Hadleyverse
  - dplyr
  - tidyr
  - ggplot2
- Useful packages for air toxics
  -raqdm
  - rucl
  - openair
- Air toxics analysis demo
- Interactive web app with shiny
What is R and RStudio
The R Project

- R is free and open-source software for statistical computing and graphics
- Download here
- It has been very popular in academia for more than a decade
- Statistical software that can do complex analyses using built-in functions, similar to SAS
- Also a programming language that is extendable (i.e. you can write your own functions/software)
Popularity

- Originally written by statisticians for statisticians
- Now widely used in academia (not just stats departments)
- Making headway into government and industry, especially the biotech and finance sectors.
- Here is a link describing the increasing popularity of R.
After you download R, you may want to use an integrated development environment (IDE) like RStudio.

- An IDE makes R a little more user friendly.
- RStudio is free and can be downloaded at rstudio.com.
Serialization Interface for Single Objects

Description

Functions to write a single R object to a file, and to restore it.

Usage

saveRDS(object, file = "", pretty = FALSE, version = NULL,
compress = TRUE, encodex = NULL)

restoreRDS(file, refbook = NULL)

Arguments

object 

Object to serialize.

file 

A connection or the name of the file when the R object is saved to.

pretty 

Logical: if TRUE or NA, an ASCII representation is written; otherwise
(default), a binary one is used. See the comments in the help for
save.

version 

The workspace format version to use. Use "" to specify the current
default version (3). Versions prior to 2 are not supported, so this will
only be relevant when there are later versions.

compress 

Logical specifying whether saving to a named file is to use "gz" or
"zip" compression, or one of "xz", "xz12" or "xz2" to indicate the
type of compression to be used (ignore if file is a connection)
Training Outline

- What is R and RStudio
- R basics
- The Hadleyverse
  - dplyr
  - tidyr
  - ggplot2
- Useful packages for air toxics
  -raqdm
  - rucl
  - openair
- Air toxics analysis demo
- Interactive web app with shiny
R Basics - Command line

- R is a language and an environment
- You type in the “command line” and hit enter to do something, in contrast to pointing and clicking to do something
R Basics - Command line

- In this presentation, command line code will be shown in blocks with gray background shading.
- The output will be shown in a separate block below the command line input with a lighter shadow and `##` at the beginning of each line.

```
"command line"

## [1] "command line"

1 + 1

## [1] 2
```
R Basics - Math

As we demonstrated in the previous slide, you can use R as a simple calculator

3 - 1

## [1] 2

2 * 2

## [1] 4

10 / 5

## [1] 2

log(10)

## [1] 2.302585
R Basics - Variables

We can store numbers and text in variables

```r
x <- 1
x
## [1] 1

x <- 10
x
## [1] 10

y <- 2
x * y
## [1] 20
```
R Basics - Variables

```r
pollutant <- "benzene"
pollutant

## [1] "benzene"

pollutant <- "toluene"
pollutant

## [1] "toluene"
```
A series of numbers or text values and be stored together in a vector. The container for a vector is `c( )`

```r
c(1, 2, 3)
```

```
## [1] 1 2 3
```

```r
x <- c(5, 6, 7)
x
```

```
## [1] 5 6 7
```

```r
z <- c("mercury", "cadmium")
z
```

```
## [1] "mercury" "cadmium"
```
R Basics - Functions

- Variables and vectors contain data
- Functions do stuff to data
- You recognize a function by parentheses ( )

```r
x <- c(1, 2, 3, 3, 4, 5, 5, 6, 7, 7, 7)
mean(x)
```

```r
## [1] 4.545455
```

```r
median(x)
```

```r
## [1] 5
```
R Basics - Functions

- To find out how to use a function, type `?` then the function name.
- Below is an image of what you see in RStudio if you type `?mean` and hit return.
A data frame is a collection of vectors with the same length

- Basically a spreadsheet

```r
pollutant <- c("benzene", "acrolein")
value <- c(.7, .8)
unit <- c("ug/m^3", "ug/m^3")
df <- cbind(pollutant, value, unit)
df

# pollutant  value  unit
# [1,] "benzene" "0.7" "ug/m^3"
# [2,] "acrolein" "0.8" "ug/m^3"
```
Those are the basic type of objects you’ll need to be familiar with to understand this training/demonstration.

Vectors and data frames store numbers and text (a variable with a single value is actually just a vector of length 1).

Functions do things to vectors and data frames then return an output.
R Basics - Installing/loading packages

- R comes with functions immediately available
- Some functions come with R, but they are only available after loading a package using library()

```r
library(lattice)
```

- Some packages do not come with an R download
- Packages that are on CRAN can be downloaded and installed using install.packages()

```r
install.packages("devtools")
```
R Basics - Installing from GitHub

- You may want to use a package that isn’t on CRAN
- Many R developers put their packages on GitHub
R Basics - Installing from GitHub

To install a package from GitHub, load the devtools package

```r
library(devtools)
```

Then use the `install_github()` function by specifying the user name and the repository

```r
install_github("ramnathv/rCharts")
```
R Basics - Resources

- Many great resources for learning R
- Beginners material
  - Quick R
  - UCLA
  - DataCamp
  - Code School
- Intermediate/advanced material
  - Cookbook for R
  - Coursera
  - Advanced R
Training Outline

- What is R and RStudio
- R basics
- The Hadleyverse
  - dplyr
  - tidyr
  - ggplot2
- Useful packages for air toxics
  -raqdm
  - rucr
  - openair
- Air toxics analysis demo
- Interactive web app with shiny
The Hadleyverse

- The Hadleyverse is a collection of packages written by Hadley Wickham
- These packages make R much easier to use
- We will introduce three useful packages for data manipulation (dplyr), reshaping data (tidyr), and data visualization (ggplot2)
- Image credit
Data manipulation with dplyr
dplyr - Overview

- A very handy data manipulation package
- Very fast (much of it is written in C++)
- Can work with tables from databases
- Main functions are `filter()`, `summarize()`, and `mutate()`
- These main functions take a data frame as input and return a data frame as output
Below is a *made up* data frame that will be used to easily visualize `dplyr` operations:

```r
tox <- read.table(header=T, text='
  monitor parameter    day   hour value screen
A   benzene  20140601 08:00  0.72  1.3
A   benzene  20140601 09:00  0.84  1.3
A   benzene  20140602 08:00  1.35  1.3
A   benzene  20140602 09:00  0.94  1.3
B   benzene  20140601 08:00  0.61  1.3
B   benzene  20140601 09:00  0.70  1.3
B   benzene  20140602 08:00  0.99  1.3
B   benzene  20140602 09:00  1.70  1.3
A   toluene  20140601 08:00  6.10 300
A   toluene  20140601 09:00  2.51 300
')
```
The first argument of filter() is a data frame, `filter(tox, ...)`
- The data frame can then be filtered using logical expressions
- Here we filter the tox data frame so that we just get values from monitor A

```r
library(dplyr)
filter(tox, monitor == "A")
```

<table>
<thead>
<tr>
<th></th>
<th>monitor</th>
<th>parameter</th>
<th>day</th>
<th>hour</th>
<th>value</th>
<th>screen</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>benzene</td>
<td>20140601</td>
<td>08:00</td>
<td>0.72</td>
<td>1.3</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>benzene</td>
<td>20140601</td>
<td>09:00</td>
<td>0.84</td>
<td>1.3</td>
</tr>
<tr>
<td>3</td>
<td>A</td>
<td>benzene</td>
<td>20140602</td>
<td>08:00</td>
<td>1.35</td>
<td>1.3</td>
</tr>
<tr>
<td>4</td>
<td>A</td>
<td>benzene</td>
<td>20140602</td>
<td>09:00</td>
<td>0.94</td>
<td>1.3</td>
</tr>
<tr>
<td>5</td>
<td>A</td>
<td>toluene</td>
<td>20140601</td>
<td>08:00</td>
<td>6.10</td>
<td>300.0</td>
</tr>
<tr>
<td>6</td>
<td>A</td>
<td>toluene</td>
<td>20140601</td>
<td>09:00</td>
<td>2.51</td>
<td>300.0</td>
</tr>
</tbody>
</table>
You can enter more than one logical expression separated by a comma:

```r
filter(tox, monitor == "B", value > screen)
```

```
# monitor parameter       day     hour  value  screen
# 1         B  benzene  20140602 09:00 1.7   1.3
```

This is equivalent to using the `&` operator:

```r
filter(tox, monitor == "B" & value > screen)
```

```
# monitor parameter       day     hour  value  screen
# 1         B  benzene  20140602 09:00 1.7   1.3
```
The first argument of `summarize()` is also a data frame
However, the input data frame requires grouping information
For example, if we wanted to summarize the tox data frame by returning daily averages for each pollutant, we need to group the data frame by monitor, parameter, and day

```
head(tox)
```

```
##   monitor parameter day     hour value screen
## 1     A     benzene 20140601 08:00  0.72    1.3
## 2     A     benzene 20140601 09:00  0.84    1.3
## 3     A     benzene 20140602 08:00  1.35    1.3
## 4     A     benzene 20140602 09:00  0.94    1.3
## 5     B     benzene 20140601 08:00  0.61    1.3
## 6     B     benzene 20140601 09:00  0.70    1.3
```
We add grouping information by using the `group_by()` function:

```r
tox_daily <- group_by(tox, monitor, parameter, day)
```

Now we feed this data frame to the summary function,

```r
summarize(tox_daily, ...)
```

To get daily averages, we also supply a function that will return the mean, and we name it `daily_mean` (obviously not a true daily mean, with only a few hours...)

```r
summarize(tox_daily, daily_mean = mean(value))
```
## Source: local data frame [5 x 4]
## Groups: monitor, parameter [?]
##
## monitor parameter  day    daily_mean
## (fctr)  (fctr)  (int)   (dbl)
## 1      A   benzene 20140601 0.780
## 2      A   benzene 20140602 1.145
## 3      A   toluene 20140601 4.305
## 4      B   benzene 20140601 0.655
## 5      B   benzene 20140602 1.345
You can add new columns to a data frame by using the `mutate()` function.

- The output will have the same number of rows.
- The first argument is the input data frame and the subsequent arguments must be functions that return a single value for each row (i.e. not a summary function).

```r
mutate(tox, above_screen = value > screen)
```
dplyr - Add Columns

```r
mutate(tox, above_screen = value > screen)
```

<table>
<thead>
<tr>
<th></th>
<th>monitor</th>
<th>parameter</th>
<th>day</th>
<th>hour</th>
<th>value</th>
<th>screen</th>
<th>above_screen</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>benzene</td>
<td>20140601</td>
<td>08:00</td>
<td>0.72</td>
<td>1.3</td>
<td>FALSE</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>benzene</td>
<td>20140601</td>
<td>09:00</td>
<td>0.84</td>
<td>1.3</td>
<td>FALSE</td>
</tr>
<tr>
<td>3</td>
<td>A</td>
<td>benzene</td>
<td>20140602</td>
<td>08:00</td>
<td>1.35</td>
<td>1.3</td>
<td>TRUE</td>
</tr>
<tr>
<td>4</td>
<td>A</td>
<td>benzene</td>
<td>20140602</td>
<td>09:00</td>
<td>0.94</td>
<td>1.3</td>
<td>FALSE</td>
</tr>
<tr>
<td>5</td>
<td>B</td>
<td>benzene</td>
<td>20140601</td>
<td>08:00</td>
<td>0.61</td>
<td>1.3</td>
<td>FALSE</td>
</tr>
<tr>
<td>6</td>
<td>B</td>
<td>benzene</td>
<td>20140601</td>
<td>09:00</td>
<td>0.70</td>
<td>1.3</td>
<td>FALSE</td>
</tr>
<tr>
<td>7</td>
<td>B</td>
<td>benzene</td>
<td>20140602</td>
<td>08:00</td>
<td>0.99</td>
<td>1.3</td>
<td>FALSE</td>
</tr>
<tr>
<td>8</td>
<td>B</td>
<td>benzene</td>
<td>20140602</td>
<td>09:00</td>
<td>1.70</td>
<td>1.3</td>
<td>TRUE</td>
</tr>
<tr>
<td>9</td>
<td>A</td>
<td>toluene</td>
<td>20140601</td>
<td>08:00</td>
<td>6.10</td>
<td>300.0</td>
<td>FALSE</td>
</tr>
<tr>
<td>10</td>
<td>A</td>
<td>toluene</td>
<td>20140601</td>
<td>09:00</td>
<td>2.51</td>
<td>300.0</td>
<td>FALSE</td>
</tr>
</tbody>
</table>
dplyr - Add Columns

Here we add more than one column at a time

```r
mutate(tox, above_screen = value > screen,
       screen_diff = value - screen)
```

<table>
<thead>
<tr>
<th>##</th>
<th>monitor</th>
<th>parameter</th>
<th>day</th>
<th>hour</th>
<th>value</th>
<th>screen</th>
<th>above_screen</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>benzene</td>
<td>20140601</td>
<td>08:00</td>
<td>0.72</td>
<td>1.3</td>
<td>FALSE</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>benzene</td>
<td>20140601</td>
<td>09:00</td>
<td>0.84</td>
<td>1.3</td>
<td>FALSE</td>
</tr>
<tr>
<td>3</td>
<td>A</td>
<td>benzene</td>
<td>20140602</td>
<td>08:00</td>
<td>1.35</td>
<td>1.3</td>
<td>TRUE</td>
</tr>
<tr>
<td>4</td>
<td>A</td>
<td>benzene</td>
<td>20140602</td>
<td>09:00</td>
<td>0.94</td>
<td>1.3</td>
<td>FALSE</td>
</tr>
<tr>
<td>5</td>
<td>B</td>
<td>benzene</td>
<td>20140601</td>
<td>08:00</td>
<td>0.61</td>
<td>1.3</td>
<td>FALSE</td>
</tr>
<tr>
<td>6</td>
<td>B</td>
<td>benzene</td>
<td>20140601</td>
<td>09:00</td>
<td>0.70</td>
<td>1.3</td>
<td>FALSE</td>
</tr>
<tr>
<td>7</td>
<td>B</td>
<td>benzene</td>
<td>20140602</td>
<td>08:00</td>
<td>0.99</td>
<td>1.3</td>
<td>FALSE</td>
</tr>
<tr>
<td>8</td>
<td>B</td>
<td>benzene</td>
<td>20140602</td>
<td>09:00</td>
<td>1.70</td>
<td>1.3</td>
<td>TRUE</td>
</tr>
<tr>
<td>9</td>
<td>A</td>
<td>toluene</td>
<td>20140601</td>
<td>08:00</td>
<td>6.10</td>
<td>300.0</td>
<td>FALSE</td>
</tr>
<tr>
<td>10</td>
<td>A</td>
<td>toluene</td>
<td>20140601</td>
<td>09:00</td>
<td>2.51</td>
<td>300.0</td>
<td>FALSE</td>
</tr>
</tbody>
</table>
Reshape data with tidyr
tidyr - Overview

- tidyr is a very simple but very useful package
- Just does two things
  - reshapes wide to long
  - reshapes long to wide
tidyr - Long to Wide

- The tox data frame is in a long format
- You can recognize a long format if there is only one column with values

```r
head(tox)
```

<table>
<thead>
<tr>
<th></th>
<th>monitor</th>
<th>parameter</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>benzene</td>
<td>20140601</td>
<td>08:00</td>
<td>0.72</td>
<td>1.3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>benzene</td>
<td>20140601</td>
<td>09:00</td>
<td>0.84</td>
<td>1.3</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>A</td>
<td>benzene</td>
<td>20140602</td>
<td>08:00</td>
<td>1.35</td>
<td>1.3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>A</td>
<td>benzene</td>
<td>20140602</td>
<td>09:00</td>
<td>0.94</td>
<td>1.3</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>B</td>
<td>benzene</td>
<td>20140601</td>
<td>08:00</td>
<td>0.61</td>
<td>1.3</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>B</td>
<td>benzene</td>
<td>20140601</td>
<td>09:00</td>
<td>0.70</td>
<td>1.3</td>
<td></td>
</tr>
</tbody>
</table>
tidyr - Long to Wide

- The `spread()` function takes a long data frame and spreads it out into a wide data frame
- The first argument is the long data frame
- The second argument is the column name of the key
- The third argument is the column name of the values

```r
library(tidyr)

tox_wide <- spread(tox, key = parameter, value = value)
```
### tox_wide

<table>
<thead>
<tr>
<th>##</th>
<th>monitor</th>
<th>day</th>
<th>hour</th>
<th>screen</th>
<th>benzene</th>
<th>toluene</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>20140601</td>
<td>08:00</td>
<td>1.3</td>
<td>0.72</td>
<td>NA</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>20140601</td>
<td>08:00</td>
<td>300.0</td>
<td>NA</td>
<td>6.10</td>
</tr>
<tr>
<td>3</td>
<td>A</td>
<td>20140601</td>
<td>09:00</td>
<td>1.3</td>
<td>0.84</td>
<td>NA</td>
</tr>
<tr>
<td>4</td>
<td>A</td>
<td>20140601</td>
<td>09:00</td>
<td>300.0</td>
<td>NA</td>
<td>2.51</td>
</tr>
<tr>
<td>5</td>
<td>A</td>
<td>20140602</td>
<td>08:00</td>
<td>1.3</td>
<td>1.35</td>
<td>NA</td>
</tr>
<tr>
<td>6</td>
<td>A</td>
<td>20140602</td>
<td>09:00</td>
<td>1.3</td>
<td>0.94</td>
<td>NA</td>
</tr>
<tr>
<td>7</td>
<td>B</td>
<td>20140601</td>
<td>08:00</td>
<td>1.3</td>
<td>0.61</td>
<td>NA</td>
</tr>
<tr>
<td>8</td>
<td>B</td>
<td>20140601</td>
<td>09:00</td>
<td>1.3</td>
<td>0.70</td>
<td>NA</td>
</tr>
<tr>
<td>9</td>
<td>B</td>
<td>20140602</td>
<td>08:00</td>
<td>1.3</td>
<td>0.99</td>
<td>NA</td>
</tr>
<tr>
<td>10</td>
<td>B</td>
<td>20140602</td>
<td>09:00</td>
<td>1.3</td>
<td>1.70</td>
<td>NA</td>
</tr>
</tbody>
</table>
tidyr - Long to Wide

We could also spread the data frame by monitor

```r
spread(tox, key = monitor, value = value)
```

```plaintext
## parameter     day   hour  screen  A   B
## 1   benzene 20140601 08:00 1.3 0.72 0.61
## 2   benzene 20140601 09:00 1.3 0.84 0.70
## 3   benzene 20140602 08:00 1.3 1.35 0.99
## 4   benzene 20140602 09:00 1.3 0.94 1.70
## 5   toluene 20140601 08:00 300.0 6.10 NA
## 6   toluene 20140601 09:00 300.0 2.51 NA
```
The `gather()` function takes a wide data frame and gathers the columns into a long data frame.
- The first argument is the wide data frame.
- The second argument is the name of the key that will be created.
- The third argument is the name of the value column that will be created.
- The remaining arguments are the names of the columns to be gathered.
### tidyr - Wide to Long

#### head(tox_wide)

```r
## monitor  day    hour screen benzene  toluene
## 1   A  20140601 08:00 1.3   0.72     NA
## 2   A  20140601 08:00 300.0 NA    6.10
## 3   A  20140601 09:00 1.3   0.84     NA
## 4   A  20140601 09:00 300.0 NA    2.51
## 5   A  20140602 08:00 1.3   1.35     NA
## 6   A  20140602 09:00 1.3   0.94     NA
```

```r
tox_long <- gather(tox_wide, key = pollutant, value = sample_measurement, benzene, toluene)
```
## monitor day hour screen pollutant sample_measurement
## 1 A 20140601 08:00 1.3 benzene 0.72
## 2 A 20140601 08:00 300.0 benzene NA
## 3 A 20140601 09:00 1.3 benzene 0.84
## 4 A 20140601 09:00 300.0 benzene NA
## 5 A 20140602 08:00 1.3 benzene 1.35
## 6 A 20140602 09:00 1.3 benzene 0.94
## 7 B 20140601 08:00 1.3 benzene 0.61
## 8 B 20140601 09:00 1.3 benzene 0.70
## 9 B 20140602 08:00 1.3 benzene 0.99
## 10 B 20140602 09:00 1.3 benzene 1.70
## 11 A 20140601 08:00 1.3 toluene NA
## 12 A 20140601 08:00 300.0 toluene 6.10
## 13 A 20140601 09:00 1.3 toluene NA
## 14 A 20140601 09:00 300.0 toluene 2.51
## 15 A 20140602 08:00 1.3 toluene NA
## 16 A 20140602 09:00 1.3 toluene NA
## 17 B 20140601 08:00 1.3 toluene NA
Visualize data with ggplot2
ggplot2 - Overview

- Base R does have graphing functions for common plots, such as scatter plots, line graphs, box plots, and histograms
- ggplot2 makes it easier to create multi-panel/complex plots
For demonstration we’ll use the following randomly generated data:

```r
gtox <- data.frame(
  date = rep(seq(as.Date("2014-01-01"), by = 6, length = 60), 2),
  monitor = rep(c("A", "B"), each = 60),
  benzene = rgamma(120, shape = 1.5, scale = 3),
  formaldehyde = rgamma(120, shape = 1, scale = 2),
  acrolein = rgamma(120, shape = 1.5, scale = 2.5),
  p_dichlorobenzene = rgamma(120, shape = 1, scale = 3)
)

ead(rtox, 3)
```

```
##     date monitor benzene formaldehyde acrolein p_dichlorobenzene
## 1 2014-01-01    A   4.798135    0.9994482    3.647809       5.207415
## 2 2014-01-07    A   3.360651    0.0396585    3.043189       4.215365
## 3 2014-01-13    A   3.341292    3.3611858    2.503075       8.096374
```
```r
rtox_long <- gather(rtox, parameter, value = sample, benzene:p_dichlorobenzene)
head(rtox_long)
```

```
##     date   monitor parameter      sample
## 1 2014-01-01       A     benzene 4.798135
## 2 2014-01-07       A     benzene 3.360651
## 3 2014-01-13       A     benzene 3.341292
## 4 2014-01-19       A     benzene 2.097056
## 5 2014-01-25       A     benzene 9.457852
## 6 2014-01-31       A     benzene 5.969694
```
Here is code to make a time series plot of benzene at site A

- The `aes()` parameter is where we specify the x and y axes
- `geom_line()` makes it a line graph

```r
library(ggplot2)

benzene_A <- filter(rtox_long, parameter == "benzene", monitor == "A")

ggplot(benzene_A, aes(x=date, y=sample)) + geom_line() + ggtitle("Benzene Time Series 2014")
```
ggplot2 - Time Series

Benzene Time Series 2014


date

sample
Here we make a time series plot of all pollutants at monitor B

```r
site_B <- filter(rtox_long, monitor == "B")

ggplot(site_B, aes(x=date, y=sample, color=parameter)) +
    geom_line() +
    ggttitle("Toxics Time Series Monitor B 2014")
```
ggplot2 - Time Series

Toxics Time Series Monitor B 2014

- benzene
- formaldehyde
- acrolein
- p_dichlorobenzene
Here we plot all toxics data for both monitors

```r
ggplot(rtox_long, aes(x=date, y=sample, color=parameter)) + geom_line() + facet_grid(~ monitor) + ggtitle("Toxics Time Series 2014")
```
ggplot2 - Time Series

Toxics Time Series 2014

Date: Jan 2014 to Jan 2015

Parameters: benzene, formaldehyde, acrolein, p_dichlorobenzene
ggplot2 - Boxplots

```r
ggplot(rtox, aes(x=monitor, y=benzene)) + geom_boxplot()
```
ggplot2 - Boxplots

```r
ggplot(site_B, aes(x=parameter, y=sample)) + geom_boxplot()
```
ggplot2 - Boxplots

```r
ggplot(rtox_long, aes(x=parameter, y=sample)) + geom_boxplot() + facet_grid(~ monitor)
```
ggplot2 - Histograms

```
ggplot(benzene_A, aes(x=sample)) + geom_histogram()
```
ggplot2 - Histograms

```r
ggplot(site_B, aes(x=sample, fill=parameter)) + geom_histogram()
```
ggplot2 - Histograms

ggplot(rtox_long, aes(x=sample, fill=parameter)) + geom_histogram() + facet_grid(~ monitor)
Training Outline

- What is R and RStudio
- R basics
- The Hadleyverse
  - dplyr
  - tidyr
  - ggplot2
- Useful packages for air toxics
  -raqdm
  - rucl
  - openair
- Air toxics analysis demo
- Interactive web app with shiny
Air Toxics related R Packages
**raqdm - Overview**

- Provides convenient access to US EPA’s AQS Data Mart in R
- Makes use of the API discussed at [http://www3.epa.gov/airdata/toc.html](http://www3.epa.gov/airdata/toc.html)
- Additional details and sourcecode on github ([https://github.com/eballey78/raqdm](https://github.com/eballey78/raqdm))
- Still under development - Comments and Suggestions Welcome ([eb11307@gmail.com](mailto:eb11307@gmail.com))
raqdm - Features

- Query AQS Data Mart from the R console or through a convenient GUI
- Save your most common parameter options so you don’t have to enter them repeatedly
- Access to all options available in EPA’s web interface
- Import requested data directly into an R data.frame for further analysis
- Can do both synchronous and asynchronous data pulls from the AQS Data Mart
raqdm - Installation

- To install raqdm you will need the devtools package available on CRAN:

```
install.packages("devtools")
```

- With devtools installed, install raqdm with the install_github function:

```
library(devtools)
install_github("ebailey78/raqdm")
library(raqdm)
```
raqdm - Setup

- You will need a username and password from EPA to access the data
- Request username and password from EPA by emailing aqsdatamart@epa.gov
- Once you have user credentials you can save them in raqdm with the `setAQDMuser` function

```r
setAQDMuser("me@mystate.gov", "my_password", save = TRUE)
```

- You can set other default parameters with the `setAQDMdefaults` function

```r
setAQDMdefaults(state = "18", bdate = "20140101", edate = "20141231")
```

- Setting `save = TRUE` will cause defaults to be saved across R sessions
raqdm - Requesting Data

- We’ve set defaults for Indiana (18), and bdate(20140101) and edate(20141231) in the previous slide
- Now we can request 2014 benzene data from Indiana with

```r
benz_req <- getAQDMdata(param = "45201", synchronous = FALSE)
# Request benzene
```

- Or we can request all available met data with

```r
met_req <- getAQDMdata(pc = "MET", synchronous = FALSE)
```
Once the requests are processed on the server, we read in the data.

```r
benz <- getAQDMrequest(benz_req)
met <- getAQDMrequest(met_req)
```
Exposure Estimates with rucl
rucl Overview

- rucl is an R package that assists in calculating Upper Confidence Limits of the Mean (UCLs) in R
- Based heavily on U.S. EPA's ProUCL software version 4.1. ([http://www2.epa.gov/land-research/proucl-software](http://www2.epa.gov/land-research/proucl-software))
- Needs additional development and testing before official release
- Not well documented yet
rucl Features

- Test for normal, lognormal, or gamma distributions
- Handles censored and uncensored datasets
- Can calculate over 30 different UCLs
- Will recommend UCLs based on data characteristics (experimental)
rucl Installation

- To install rucl you will need the devtools package available on CRAN:

  ```r
  install.packages("devtools")
  ```

- With devtools installed, install raqdm with the install_github function:

  ```r
  library(devtools)
  install_github("ebailey78/rucl")
  library(rucl)
  ```
rucl Usage

- The primary function in rucl is ucl()
- The only required argument is a numeric vector that represents concentrations

```r
# Create a random dataset with 50 values
# from a gamma distribution
x <- rgamma(50, 5, 2)
head(x)


# Return the recommended UCL
ucl(x)

## n.tucl  n.modt
## 2.832101 2.832155
```
rucl Usage

- You can also request specific UCL calculations

```
# Calculate the modified-t-based UCL for normal distributions
ucl(x, "n.modt")
```

```
## n.modt
## 2.832155
```
**rucl Usage**

- Arguments can also be used to change the confidence level and number of bootstrap iterations

```r
ucl(x, confidence = 0.95, N = 10000)
```

```r
## n.tucl  n.modt
## 2.832101 2.832155
```

```r
ucl(x, confidence = 0.9, N = 10000)
```

```r
## n.tucl  n.modt
## 2.774649 2.774703
```
rucl Usage

- type = detailed will create a rucl object that contains a great deal of information about the dataset

```r
ucl(x, type = "detailed")
```
rucl Usage

Summary for uncensored dataset 'x'

Confidence Coefficient: 95%
Bootstrap Operations(N): 2000

Summary Statistics

<table>
<thead>
<tr>
<th></th>
<th>Minimum Value of Log Data: -0.916</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample Size:</td>
<td>50</td>
</tr>
<tr>
<td>Minimum Value:</td>
<td>0.4</td>
</tr>
<tr>
<td>Maximum Value:</td>
<td>7.46</td>
</tr>
<tr>
<td>Arithmetic Mean:</td>
<td>1.73</td>
</tr>
<tr>
<td>Geometric Mean:</td>
<td>1.16</td>
</tr>
<tr>
<td>Median:</td>
<td>1.05</td>
</tr>
<tr>
<td>Standard Deviation:</td>
<td>1.77</td>
</tr>
<tr>
<td>Standard Error:</td>
<td>0.25</td>
</tr>
<tr>
<td>Coefficient of Variance:</td>
<td>1.02</td>
</tr>
<tr>
<td>Skewness:</td>
<td>2.03</td>
</tr>
<tr>
<td>MLE of Gamma Shape:</td>
<td>1.32</td>
</tr>
<tr>
<td>MLE of Gamma Scale:</td>
<td>1.31</td>
</tr>
<tr>
<td>MLE of Gamma Rate:</td>
<td>0.761</td>
</tr>
<tr>
<td>MLE of Gamma Mean:</td>
<td>1.73</td>
</tr>
<tr>
<td>MLE of Gamma SSD:</td>
<td>1.51</td>
</tr>
</tbody>
</table>

Distribution Testing

Normal: FALSE
Lognormal: TRUE
Gamma: FALSE

Recommendation: Lognormal

Upper Confidence Limits of the Mean (UCLs)

Student's t UCL: 2.15
Adjusted Central Limit Theorem UCL: 2.22
Modified Student's t UCL: 2.15
Land's H UCL: 2.25
95% Chebyshev UCL (MVUE): 2.74
97.5% Chebyshev UCL (MVUE): 3.19
99% Chebyshev UCL (MVUE): 4.08
Approximate Gamma UCL: 2.15
Adjusted Gamma UCL: 2.16
Central Limit Theorem UCL: 2.15
Standard Bootstrap UCL: 2.13
Bootstrap t UCL: 2.28
Hall's Bootstrap UCL: 2.24
Simple Percentile Bootstrap UCL: 2.15
BCA Percentile Bootstrap: 2.2
95% Chebyshev UCL (mean, sd): 2.83
97.5% Chebyshev UCL (mean, sd): 3.3
99% Chebyshev UCL (mean, sd): 4.23
rucl Censored Data

- rucl can handle censored datasets using Kaplan-Meier
- Requires a second vector with TRUE/FALSE indicating whether the corresponding reading is a detect (TRUE) or nondetect (FALSE)
- For non-detects, assumes reported reading is detection limit

```r
# Create a lognormal dataset with 50 readings censored at 0.4
x <- rlnorm(50)
d <- x > 0.4
x[!d] <- 0.4

ucl(x, d = d)
```
rucl Censored Data

## o.km.cheb95
## 2.842027
Data visualization with openair
The openair project

- The openair package was created by Dr. David Carslaw at King’s College London.
- The goals of the openair project are to create tools in R that use the wealth of air pollution data that is publicly available to make it easy to carry out sophisticated analyses quickly and in a reproducible way.
- Allow the user to quickly summarize and visualize air pollution data through the use of wind and pollution roses, trend analyses, and other helpful tools.
- Comprehensive user manual can be found here: (http://www.openair-project.org/downloads/openairmanual.aspx)
The first thing we want to do after importing our data is a quick summary plot.

- Creates time-series plots and makes it easy to see chunks of missing data.
- The summary plot also displays basic summary stats such as mean, median and the 95th percentile.
- A histogram helps the user view the distributions of each of their parameters.
- You can call this plot using the following code:

```r
library(openair)
summaryPlot(mydata)
```
Summary Plot

```
## date1 date2 nox no2 co
## "POSIXt" "POSIXct" "integer" "integer" "numeric"
```

![Summary Plot Diagram](image)

- **NO**:
  - Missing values: 7033 (10%)
  - Minimum: 0
  - Maximum: 206
  - Mean: 49.1
  - Median: 46
  - 95th Percentile: 93

- **NO2**:
  - Missing values: 6531 (9.3%)
  - Minimum: 0
  - Maximum: 19.7
  - Mean: 1.5
  - Median: 1.1
  - 95th Percentile: 3.7

- **CO**:
  - Missing values: 7018 (10%)
  - Minimum: 0
  - Maximum: 1144
  - Mean: 178.8
  - Median: 153
  - 95th Percentile: 414
Windrose

- The windrose plot allows easy visualization of wind speed and wind direction data.
- Could be especially useful for a modeler or analyst who wants to look at many years or different seasons of wind data at a monitor.
Windrose

\texttt{windRose(mydata, type="season", paddle=F)}

- **Spring (MAM)**: Mean = 4.47, Calm = 0.1%
- **Summer (JJA)**: Mean = 4.22, Calm = 0%
- **Autumn (SON)**: Mean = 4.33, Calm = 0.1%
- **Winter (DJF)**: Mean = 4.93, Calm = 0.1%

Frequency of counts by wind direction (%)

- 0 to 2 m s\(^{-1}\)
- 2 to 4 m s\(^{-1}\)
- 4 to 6 m s\(^{-1}\)
- 6 to 20.16 m s\(^{-1}\)
smoothTrend(\texttt{mydata}, \texttt{pollutant = "no2"}, \texttt{deseason = TRUE}, \texttt{simulate = TRUE}, \texttt{ylab = "concentration (ppb)"}, \texttt{main = "monthly mean deseasonalized no2"})
Training Outline

- What is R and RStudio
- R basics
- The Hadleyverse
  - dplyr
  - tidyr
  - ggplot2
- Useful packages for air toxics
  - raqdm
  - rucl
  - openair
- Air toxics analysis demo
- Interactive web app with shiny
Getting the data

- Use raqdm to retrieve 2014 core HAPs data for the Gary IITRI monitor in Lake County, Indiana
- Below is another example of an asynchronous AQDM data pull

```r
haps_request <- getAQDMdata(state = "18", county = "089", site = "0022", pc = "CORE_HAPS", format = "DMCSV", dur = "7", bdate = "20140101", edate = "20141231")

haps <- getAQDMrequest(haps_request)
```
Summarizing data with dplyr

- Group the data using dplyr’s `group_by` function to group the data by parameter

```r
haps <- group_by(haps, AQS.Parameter.Desc)
```

- Then summarize the data in a table

```r
haps_summary <- summarize(haps, n = n(),
                          max = max(Sample.Measurement),
                          median = median(Sample.Measurement),
                          mean = signif(mean(Sample.Measurement), 2),
                          st.dev = signif(sd(Sample.Measurement), 2),
                          ucl = signif(max(ucl(Sample.Measurement)), 2))
# from rucl package

# Remove pollutants with no detectable readings
haps_summary <- haps_summary[haps_summary$max != 0, ]
```
Summarizing data with dplyr

```r
head(haps_summary, n=8)
```

```r
## Source: local data frame [8 x 7]
##
## AQS.Parameter.Desc (fctr) n max median mean st.dev ucl  
## (int) (dbl) (dbl) (dbl) (dbl) (dbl) (dbl)
## 1 1,3-Butadiene 55 0.300 0.000 0.06400 0.1000 0.0880
## 2 Acetaldehyde 68 2.400 1.200 1.30000 0.4100 1.4000
## 3 Acrolein - Unverified 55 2.900 1.300 1.40000 0.5800 1.6000
## 4 Arsenic PM2.5 LC 57 0.011 0.001 0.00110 0.0018 0.0026
## 5 Benzene 110 9.400 0.900 1.80000 2.1000 2.7000
## 6 Cadmium PM2.5 LC 57 0.009 0.000 0.00061 0.0017 0.0016
## 7 Carbon tetrachloride 55 0.100 0.100 0.09600 0.0190 0.1200
## 8 Chromium PM2.5 LC 57 0.013 0.000 0.00110 0.0020 0.0020
A closer look at benzene

► Use raqdm to retrieve 2013-2014 24-hour benzene data for all of Indiana

```r
benzene_request <- getAQDMdata(state = "18", param = "45201", 
format = "DMCSV", dur = "7",
    bdate = "20130101", edate = "20141231")

benzene <- getAQDMrequest(benzene_request)
```
Prepare the data

```r
# Convert from ppbc to ppbv
benzene$Sample.Measurement <- benzene$Sample.Measurement / 6

# Convert from ppbv to ug/m3
benzene$Sample.Measurement <-
    benzene$Sample.Measurement * 78.11 / 24.45

# Combine the State.Code, County.Code, and Site.Num,
# and POC fields into a single Site.ID field
benzene$Site.ID <- sprintf("%02i-%03i-%04i-%i",
    benzene$State.Code,
    benzene$County.Code,
    benzene$Site.Num,
    benzene$POC)

benzene <- group_by(benzene, Site.ID)
```
Summarize the data

```r
summarize(benzene,
    n = n(), max = max(Sample.Measurement),
    ucl = ucl(Sample.Measurement)[1],
    risk = ucl * 7.8e-6)
```

## Source: local data frame [10 x 5]

<table>
<thead>
<tr>
<th>Site.ID</th>
<th>n</th>
<th>max</th>
<th>ucl</th>
<th>risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>(chr)</td>
<td>(int)</td>
<td>(dbl)</td>
<td>(dbl)</td>
<td>(dbl)</td>
</tr>
<tr>
<td>1 18-019-0009-1</td>
<td>116</td>
<td>1.171384</td>
<td>0.4699129</td>
<td>3.665321e-06</td>
</tr>
<tr>
<td>2 18-089-0022-2</td>
<td>110</td>
<td>18.156449</td>
<td>1.8441180</td>
<td>1.438412e-05</td>
</tr>
<tr>
<td>3 18-089-0022-7</td>
<td>109</td>
<td>18.316183</td>
<td>2.0649492</td>
<td>1.610660e-05</td>
</tr>
<tr>
<td>4 18-089-0030-1</td>
<td>116</td>
<td>2.449257</td>
<td>0.7675006</td>
<td>5.986505e-06</td>
</tr>
<tr>
<td>5 18-089-0034-1</td>
<td>118</td>
<td>2.289523</td>
<td>0.7240966</td>
<td>5.647954e-06</td>
</tr>
<tr>
<td>6 18-089-2008-1</td>
<td>122</td>
<td>2.875215</td>
<td>0.8292063</td>
<td>6.467809e-06</td>
</tr>
<tr>
<td>7 18-097-0078-1</td>
<td>112</td>
<td>1.703831</td>
<td>0.7329372</td>
<td>5.716910e-06</td>
</tr>
<tr>
<td>8 18-127-0024-1</td>
<td>118</td>
<td>1.703831</td>
<td>0.6072662</td>
<td>4.736676e-06</td>
</tr>
<tr>
<td>9 18-163-0016-1</td>
<td>113</td>
<td>1.224628</td>
<td>0.5358455</td>
<td>4.179595e-06</td>
</tr>
<tr>
<td>10 18-167-0025-1</td>
<td>56</td>
<td>1.011650</td>
<td>0.5643314</td>
<td>4.401785e-06</td>
</tr>
</tbody>
</table>
Boxplots

# Create a boxplot of the benzene data
# grouped by the new Site.ID field

ggplot(benzene, aes(Site.ID, Sample.Measurement)) +
  geom_boxplot()
Line Graphs

- Let's look at benzene data at the Gary IITRI monitor over a longer time period.

```r
# Let's get the data

# Using the getAQDMdata function

gi_benz_req <- getAQDMdata(state = "18", county = "089",
                           site = "0022", param = "45201",
                           format = "DMCSV", dur = "7",
                           bdate = "20120101", edate = "20141231")

# Retrieve the data

benz <- getAQDMrequest(gi_benz_req)

# Plot the data

ggplot(benz, aes(as.Date(Date.Local), Sample.Measurement)) +
       geom_line() + scale_x_date() + xlab("") +
       ylab("Concentration")
```
Plotting Gary Benzene data with openair
Retrieving AQS data

- Before we can use openair we will review how to grab data with raqdm and reshape it with tidyr. This will get the data into a format that openair requires.

- We will use raqdm to retrieve Photochemical Assessment Monitoring Station (PAMS) data from the Gary IITRI monitor.

library(raqdm)
We will query the PAMS data from the Gary IITRI site using the following request:

```r
pams <- getAQDMdata(user = "me@email.com", pw = "abc123",
  state = "18", county = "089", site = "0022",
  pc = "PAMS", param="", format = "DMCSV",
  bdate = "20140601", edate = "20140602",
  frmonly = "n", synchronous = TRUE)
```
Asynchronous query

This previous query was only for a couple of days. If we want to query for the entire year for a whole class of parameters, then an asynchronous retrieval is the way to go.

```r
pams <- getAQDMdata(user = "me@email.com", pw = "abc123",
                      state = "18", county = "089", site = "0022",
                      pc = "PAMS", param="", format = "DMCSV",
                      bdate = "20140101", edate = "20141231",
                      frmonly = "n", synchronous = FALSE)

pams <- getAQDMrequest(pams)
```
Tidying the data for openair

As we said before we will need to use tidyr to reshape the data into a format that openair likes. openair requires data in a ‘wide’ format with all of the variables in their own column like this:

```r
library(openair)
head(mydata, n=2)
```

```
#>   date       ws  wd nox no2  o3 pm10  so2  co pm25
#> 1 1998-01-01  0.60 280 285  39  1  29  4.72 3.37  NA
#> 2 1998-01-01  2.16 230   NA   NA  NA  37   NA  NA  NA
```
Use `dplyr` to filter the data

- This returns a huge dataset (513,072 rows) which includes all the VOCs, NOx and meteorological parameters for the Gary IITRI site for 2014.
- Now let’s use `dplyr` to filter it down to just a few of the parameters that we will need for our analysis in `openair` (i.e. benzene, toluene, wind speed, and wind direction).

```r
library(dplyr)
```
Select columns of interest

Then we will select only the columns of interest to us using the `select()` function from `dplyr`. For this analysis, we only need `Date (Date.GMT), Hour(X24.Hour.GMT), Parameter(AQS.Parameter.Desc), and Sample Measurement(Sample.Measurement).

IITRI_sub <- select(IITRI_sub, Date.GMT, X24.Hour.GMT, AQS.Parameter.Desc, Sample.Measurement)
Format data

This leaves us with:

<table>
<thead>
<tr>
<th>#</th>
<th>Date.GMT</th>
<th>X24.Hour.GMT</th>
<th>AQS.Parameter.Desc</th>
<th>Sample.Measurement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2014-01-02</td>
<td>01:00</td>
<td>Wind Direction - Resultant</td>
<td>66</td>
</tr>
<tr>
<td>2</td>
<td>2014-01-03</td>
<td>01:00</td>
<td>Wind Direction - Resultant</td>
<td>330</td>
</tr>
<tr>
<td>3</td>
<td>2014-01-04</td>
<td>01:00</td>
<td>Wind Direction - Resultant</td>
<td>176</td>
</tr>
<tr>
<td>4</td>
<td>2014-01-05</td>
<td>01:00</td>
<td>Wind Direction - Resultant</td>
<td>191</td>
</tr>
<tr>
<td>5</td>
<td>2014-01-06</td>
<td>01:00</td>
<td>Wind Direction - Resultant</td>
<td>315</td>
</tr>
<tr>
<td>6</td>
<td>2014-01-07</td>
<td>01:00</td>
<td>Wind Direction - Resultant</td>
<td>253</td>
</tr>
</tbody>
</table>
Now that we have selected our columns of interest, let's use `tidyr` to spread out our 'long' data and make it 'wide'.

```r
library(tidyverse)
gary_oa <- spread(IITRI_sub, key = AQS.Parameter.Desc,
                   value = Sample.Measurement)
# Convert from ppbc to ppbv
gary_oa$Benzene <- gary_oa$Benzene / 6
gary_oa$Toluene <- gary_oa$Toluene / 7
head(gary_oa, n=1)
```

```r
## Date.GMT X24.Hour.GMT Benzene Toluene Wind Direction - Resultant Wind Speed - Resultant
## 1 2014-01-01 06:00 NA NA 38 8.1
```
Then we will write it to a csv file for easy import into openair

```r
write.csv(gary_oa,
    file = file.path(tempdir(), "Gary_openair.csv"),
    row.names=F)
```
Importing into openair

- Now our data is ready to use in openair
- We will use the import() function in openair to bring in our csv file.
- The import function is helpful because it takes care of all of the final date formatting for openair plots
- Once the data is imported into openair we can do a summaryPlot to quickly view the data

```r
library(openair)
gary <- import(file = file.path(tempdir(), "Gary_openair.csv"),
    sep=",", date="Date.GMT", time="X24.Hour.GMT",
    date.format="%Y-%m-%d", time.format="%H:%M",
    ws="Wind Speed - Resultant",
    wd="Wind Direction - Resultant")

summaryPlot(selectByDate(gary, year=2014))
```
Summary Plot

## date1 date2 X24.Hour.GMT
## "POSIXct" "POSIXt" "factor"
## wd ws
## "numeric" "numeric"

- **Benzene**
  - missing = 1759 (20.1%)
  - min = 0
  - max = 2
  - mean = 0.1
  - median = 0.1
  - 95th percentile = 0.5
  - 79.9%

- **Toluene**
  - missing = 54 (0.6%)
  - min = 1
  - max = 360
  - mean = 172.9
  - median = 186
  - 95th percentile = 328
  - 99.4%

- **wind dir.**
  - missing = 54 (0.6%)
  - min = 0.1
  - max = 27.3
  - mean = 6.5
  - median = 6.1
  - 95th percentile = 12.8
  - 99.4%

- **wind spd.**
  - missing = 54 (0.6%)
  - min = 0.1
  - max = 27.3
  - mean = 6.5
  - median = 6.1
  - 95th percentile = 12.8
  - 99.4%
A pollution rose plot is useful for describing the proportion of contaminant that comes from each wind direction.
Pollution Rose

\[\text{pollutionRose}(\text{gary, pollutant= "Benzene"})\]
Percentile Rose

- A percentile rose plot can help you see the distribution of concentrations by wind direction.
percentileRose(gary, pollutant="Benzene", percentile = c(0, 1, 50, 75, 95), smooth=TRUE)
Time-series plots

timePlot(gary, pollutant=c("Benzene", "Toluene"))
Calendar Plots

calendarPlot(gary, pollutant = "Benzene", year=2014,
statistic="mean",
labels=c("acceptable","near cancer screening level",
"above cancer screening level"),
breaks=c(0,0.4,1.4,200),
main = "2014 Gary IITRI Benzene (ppbv)"
)
Calendar Plots

2014 Gary IITRI Benzene (ppbv)

- **Acceptable**
- **Near cancer screening level**
- **Above cancer screening level**

---

- **January**
- **February**
- **March**
- **April**

---

- **May**
- **June**
- **July**
- **August**

---

- **September**
- **October**
- **November**
- **December**

---
You can use the `annotate()` function within `calendarPlot` to plot the wind direction (annotate="wd") or the concentrations (annotate = "value") on the calendar.

```r
calendarPlot(gary, pollutant = "Benzene", year=2014, 
annotate="wd", statistic="mean", 
labels=c("acceptable","near cancer screening level", 
         "above cancer screening level"), 
breaks=c(0,0.4,1.4,200), 
main = "2014 Gary IITRI Benzene (ppbv) with Wind Direction")
```
Calendar Plots

2014 Gary IITRI Benzene (ppbv) with Wind Direction

acceptable
near cancer screening level
above cancer screening level
Training Outline

- What is R and RStudio
- R basics
- The Hadleyverse
  - dplyr
  - tidyr
  - ggplot2
- Useful packages for air toxics
  -raqdm
  - rucl
  - openair
- Air toxics analysis demo
- Interactive web app with shiny
Shiny - Overview

- **shiny** is a package developed by RStudio
- Makes it easy to create web applications using the R language
- Not necessary to know HTML, CSS, JavaScript
- But you can customize your apps using those web languages
- The simplest way to create an app is to make two “R Script” files
  - **ui.R** is a file that will determine what the user interface will look like
  - **server.R** is a file that will do the data manipulation and create the output for the app
library(shiny)
shinyUI(fluidPage(
    titlePanel("Hello Shiny!"),
    sidebarLayout(
        sidebarPanel(
            sliderInput("bins",
                "Number of bins: ",
                min = 1,
                max = 50,
                value = 30)
        ),
        mainPanel(
            plotOutput("distPlot")
        )
    )
))
library(shiny)

shinyServer(function(input, output) {
  output$distPlot <- renderPlot({
    x <- faithful[, 2]  # Old Faithful Geyser data
    bins <- seq(min(x), max(x), length.out = input$bins + 1)
    hist(x, breaks = bins, col = 'darkgray', border = 'white')
  })
})
Save the `ui.R` and `server.R` files in your working directory
Also save your data in your working directory
Use `runApp()` to bring up your application in a web browser
Shiny app
library(shiny)

load("haps.rda")
pollutants <- unique(as.character(haps$AQS.Parameter.Desc))

shinyUI(fluidPage(
  titlePanel("Gary IITRI HAPS 2014"),
  sidebarLayout(
    sidebarPanel(
      selectInput("pollutants", "Choose pollutants: ",
        choices = pollutants,
        selected = "Benzene", multiple = TRUE)
    ),
    mainPanel(
      plotOutput("ggplot")
    )
  )
))
library(shiny)
library(dplyr)
library(ggplot2)

load("haps.rda")

shinyServer(function(input, output) {

  output$ggplot <- renderPlot({

    haps <- filter(haps,
                     AQS.Parameter.Desc %in% input$pollutants)

    ggplot(haps, aes(x=AQS.Parameter.Desc, y=Sample.Measurement) + geom_boxplot() + facet_grid(Unit.of.Measure ~ ., scales = "free")
  })
})
Choose pollutants:
Benzene
Contact Information

- Nathan Byers, natebyers@gmail.com
- Kali Frost, kdfrost14@gmail.com
- Eric Bailey, eb11307@gmail.com